Generative Modeling

VAEs

They aim to learn latent representations of data.

1- Encoder $q\phi(z \mid x)$ that maps input data x to
a latent space z.
2- Decoder $p\theta(x \mid z)$ that reconstructs the input data from the latent space

Objective: Minimize the KL divergence between the approximate distribution $q\phi(z \mid x)$ and the true posterior $p(z \mid x)$:
$\phi* = \arg\min\phi\ KL(q\phi(z \mid x) // p(z \mid x))$

True Posterior: $p(z \mid x)$ is the actual distribution of latent variables z given observed data x and we want to calculate that BUT $p(z \mid x)$ is intractable to calculate especially for large scale inputs because each dimension will bring an extra nested integral calculation.

Cannot imagine this for images

That is why during training we use Variational Inference and convert this non-intractable problem into a ELBO

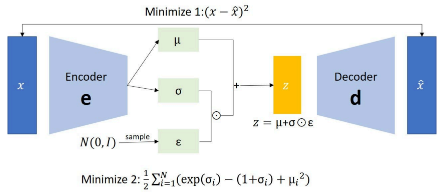
$$
\begin{aligned}
\log p(\mathbf{x}) &= \log \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})\,d\mathbf{z} \\
&= \log \int \overset{=1}{\frac{q_\phi(\mathbf{z})}{q_\phi(\mathbf{z})}} p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})\,d\mathbf{z} \\
&= \log \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z})}\right] \\
&\geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z})}\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}[\log p(\mathbf{z}) - \log q_\phi(\mathbf{z})] \\
&= \underbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction term}} - \underbrace{\mathrm{KL}[q_\phi(\mathbf{z}) \parallel p(\mathbf{z})]}_{\text{Regularization term}}
\end{aligned}
$$

It regularizes the latent space by encouraging $q\ \phi(z \mid x)$ to be close to $p(z)$, the prior distribution.

Reparametrization Trick
Here $q\phi(z)$ or $q\phi(z|x)$ is not differentible because of the randomness in sampling z.

To sample z, we would typically:
1. Draw a random variable ϵ from a standard normal distribution N(0,1).
2. Compute z using the transformation:

$z = \mu\phi(x) + \sigma\phi(x) \cdot \epsilon$
Here, ϵ is a random variable, and z depends on this randomness.
The value of z changes every time you sample a new ϵ, making it non-differentiable because the randomness introduces discontinuity.



Autoregressive

The fundamental concept involves representing a sequence of data points by defining each point as a function of the preceding points.

$$p(\mathbf{x}) = p(x_1)\prod_{d=2}^{D} p(x_d|\mathbf{x}_{<d})$$

For example: $p(\mathbf{x}) = p(x_1)\, p(x_2|x_1)\, p(x_3|x_1, x_2)\, p(x_4|x_1, x_2, x_3)$

Lets think about image generation:

x_i is the value of the i-th pixel, and $x_{1:i-1}$ are the values of all preceding pixels.

Then the loss for generating an image can look like this:

$$\mathcal{L}(x) = -\sum_{i=1}^{H}\sum_{j=1}^{W} \log p_\theta(x_{i,j}|x_{1:i-1,:}, x_{i,1:j-1})$$

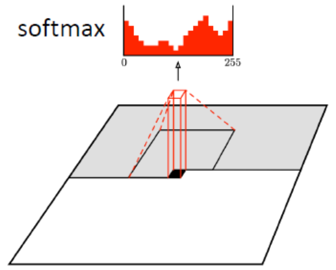
There we have to use masked convolutions because as can be seen from the formula that we cannot have access to all pixel values. In a masked convolution, we ensure that the receptive field of each pixel only includes the previously generated pixels.
For each pixel $x_{i,j}$, the conditional probability is given by $p(x_{i,j} \mid x_{1:i-1,:}, x_{i,1:j-1})$

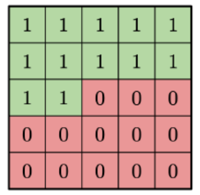
Lets visualize this with very low level example for image generation.
Assume we have image matrix like this:

$$\begin{bmatrix} \underline{100} & \underline{150} & \underline{200} \\ 50 & 125 & 175 \\ 25 & 75 & 225 \end{bmatrix}$$

The loss for construction an image would look like this:

$$\mathcal{L}(x) = -[\underline{\log p_\theta(100)} + \underline{\log p_\theta(150|100)} + \underline{\log p_\theta(200|100, 150)} + \cdots]$$





1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

Normalizing Flow

The goal of normalizing flows is to model the data distribution $p(x)$.
This is done by transforming a simple base distribution $p(z)$(typically a standard normal distribution) through a series of invertible transformations $f\theta$.

In flow models we start with simple distribution (for example Gaussian) and apply invertible and learnable flow functions $f\theta$ parameterized by θ.
Each transformation fi maps a variable zi to $zi+1$.

$f\theta = fK \circ fK-1 \circ \ldots \circ f1$

Transformed data distribution matches the original data distribution

- Apply the sequence of transformations to the data point x to get $z = f\theta(x)$
- Compute the log-likelihood of z under the base distribution $p(z)$:

$\log p(z) = \log N(z ; 0, I) = -0.5\ (zT\ z + D \log(2\pi))$ where D is the dimensionality of z.

- Then compute

$$\log\left|\det\left(\frac{\partial f_\theta(x)}{\partial x}\right)\right|$$

The total log-likelihood of the data point x is

$$\log p(x) = \log p(z) + \log\left|\det\left(\frac{\partial f_\theta(x)}{\partial x}\right)\right|$$

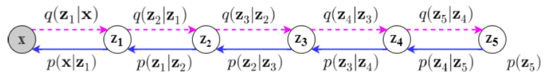
Note: The transformations we apply need to be inversible.
To say a function is inversible it has to be surjective and injective

Surjective Rule: If there is an input value for an output, it is subjective. In other words, if there is a solution for a function than it is surjective.

Injective Rule: If a function never maps to the same output of different inputs then it is injective.
$f(x1) \neq f(x2)$

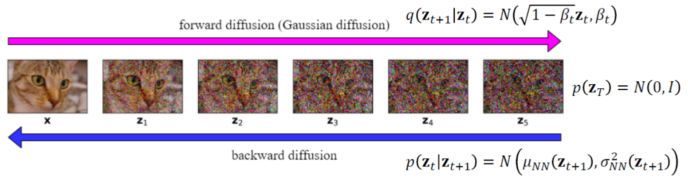
Diffusion

Composed of 2 main steps:
1. Forward Diffusion process
2. Reverse Diffusion process



Forward Diffusion process

This process adds noise to input data step by step until it becomes indistinguishable from random noise.

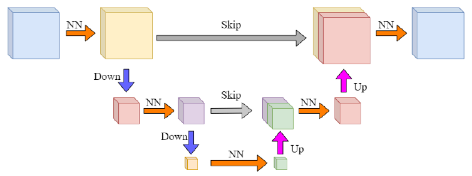


Define Gaussian Noise to the data over T timesteps.
At each timestep t, the image x is progressively corrupted:

$$x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

Reverse Diffusion process

We have neural network (generally U-Net).



The model learns to reverse the noising process.
It takes a noisy image xt and predicts the noise defined over that input.
It calculates the loss by looking real noise and predicted noise and updates the model parameters based on that.
And of course as a last step it reverses the predicted noise to generate the original input sample.